EVOLUTIONARY ALGORITHMS FOR CAPACITATED ARC ROUTING PROBLEMS WITH TIME WINDOWS

Wahiba Ramdane-Cherif

MACSI Team of LORIA-CNRS and INRIA LORRAINE
Ecole des Mines de Nancy, Parc de Saurupt, 54042, Nancy Cedex, France
ramdanec@loria.fr

Abstract: The Capacitated Arc Routing Problem (CARP) involves vehicles routing, serving a set of arcs in a network. This NP hard problem is extended to take into account time windows, entailing a new and hard theoretical model in arc routing called the CARPTW (CARP with time windows). The CARPTW is useful for modeling urban waste collection or winter gritting. This paper presents this new model and a memetic algorithm with new memetic operators able to tackle the time windows constraints in arc routing.

Keywords: Transportation, Arc Routing, Time Windows, Memetic Algorithm

1. INTRODUCTION AND BACKGROUND

Logistic can be defined as providing goods and services from a supply point to various demand ones. It requires effective transportation management, because it can save company a considerable part of its total expenses. Routing and scheduling problems are important elements of many logistic systems. The most studied routing problems, like the *VRP* (*Vehicle Routing Problem*) consist of processing demands located on the nodes of a network. When the demands are placed on the arcs, the equivalent problem is a *Capacitated Arc Routing Problem* (*CARP*). The basic *CARP* is defined in the literature on an undirected network. A fleet of identical vehicles with a fixed capacity is located at a specific node (the depot). Each edge has a non-negative demand and traversal cost. All edges with a non-zero demand must be processed. The *CARP* consists of determining a set of trips of minimum total cost, such as:

- each trip, performed by one single vehicle, starts and ends at the depot ;
- each edge with a non-zero demand can be traversed several times, by the same trip or by distinct trips, but it must be processed by one single trip, and during one traversal only;
- the sum of demands collected by a trip cannot exceed vehicle capacity.

Many applications concern road networks: collection of municipal waste, snow removal, sweeping, gritting, spraying herbicides on rails or roads to kill weeds and mail delivery. In that case, the demands generally correspond to quantities to be collected in the streets (for the municipal waste) or delivered (salt or sand for ice clearance in winter). The costs are for example distances or travel times. The undirected case concern roads that can be processed in one traversal, and in any direction. It is also possible to define a directed *CARP*, in which an arc represents one street, or one side of street with a mandatory direction for processing.

Both the undirected and directed *CARP* are NP-hard, even in the special case where only one vehicle with infinite capacity is available (*Rural Postman Problem* or *RPP*). The Chinese Postman Problem (*CPP*) is a particular case of *RPP* in which all edges have a non-zero demand. It is polynomial for directed and undirected networks, but becomes NP-hard for mixed graphs.

The existing exact methods (Hirabayashi, *et al.*, 1992) are limited to 20 edges, explaining why real cases must be tackled by heuristics like Path-Scanning (Golden, *et al.*, 1983) and Augment-Merge (Golden and Wong, 1981); or by recent metaheuristics like the tabu search method of (Hertz, *et al.*, 2000), a variable neighborhood descent method (Hertz and Mittaz, 2001), a guided local search designed by (Beullens, *et al.*, 2001) and the hybrid genetic algorithm proposed by (Lacomme, *et al.*, 2001, 2004) which is currently the most efficient solution method.
However, the academic *CARP* with an undirected graph is not realistic enough for applications such as waste collection. The undirected graph can only

model two-way streets both of whose sides can be collected in parallel and in any direction. Though this bilateral collection is common in narrow streets, real networks also include two-way streets with independent sides and one-way streets. Such networks require a mixed graph model. Of course, the *CARP* is not limited in theory to the case of edges but, to the best of our knowledge, only one paper explains how to tackle mixed graphs (Lacomme, *et al*., 2004). Recently, three new generalisations of the *CARP* were studied:

- the *ECARP* (*Extended CARP*) (Lacomme, *et al*, 2001, 2004) that copes with very realistic networks with one-way and two-way streets, prohibited turns, multiple dumping sites, etc.
- the *SCARP* (*Stochastic CARP*) (Fleury, *et al*., 2005), whose the aim is to compute robust solutions, keeping their quality in case of imprecise data.
- the *PCARP* (*Periodic CARP*) (Lacomme, *et al*., 2005) is a long-term planning problem in which treatment days must be assigned to each street, subject to given frequencies, before computing the trips for each day. The objective is to minimize a total cost over the horizon considered.

The aim of this paper is to study the *ECARP* with time windows (*CARPTW*), in order to reduce the gap between academic arc routing models and complex real applications. This new problem is useful for waste collection, mail and newspaper delivery, or inspection of power lines.

The *VRP* has already its extension to the *VRPTW* which is NP-hard in the strong sense (Kohl, 1995). Several authors suggest optimal solution methods only for small problems. Even if these methods get satisfactory results for solving small size problems, they are not efficient when the size of the problem increases. For solving more realistic cases to optimality, a lot of research based on heuristic methods has been done. Several authors have used route building heuristics. Some others have used improvement heuristics that combine route building heuristics with local search. The best results when solving realistic problems have been achieved using metaheuristic techniques: Tabu Search (Schulze and Fahle, 1999), Genetic Algorithm (Thangiah, 1995) and simulated Annealing (Chiang and Russel, 1996).

In theory, any *CARP* instance with k required edges could be converted into an equivalent *VRP* with $3k+1$ nodes, as explained by (Pearn, *et al*., 1987). This technique could be applied to *CARP* to get *VRP* but it is not used in this paper because it is no longer valid for mixed graphs and the increase in size often leads to intractable instances.

The reminder of this paper is organized as follows: section 2 recalls the memetic algorithm developed for the *ECARP* (Lacomme, *et al*., 2001). Some of its components are recycled in a non trivial way and new components are added to build a memetic algorithm for the *CARPTW*. Section 3 describes the *CARPTW*. Constructive methods are proposed in section 4 and section 5 deals with the memetic algorithm.

2. A MEMETIC ALGORITHMS FOR THE ECARP

2.1 An Extended CARP for Mixed Networks

We describe below notations and data structures for a more realistic *CARP* based on a mixed network with prohibited turns. The network must be encoded as an entirely directed graph $G = (N, A)$, with a set N of nn nodes (crossroads) and a set A of na arcs (lanes with traffic directions). N includes a depot node with nva vehicles of capacity Q (nva is either fixed or left as a decision variable). The classical notation of arcs as pairs of nodes becomes ambiguous in case of multiple lanes and prohibited turns, e.g., the shortest path from a node i to a node j depends on the arcs used to reach i and leave j. We then prefer to use arc indexes from 1 to na.

Each arc u begins at a node $b(u)$, ends at a node $e(u)$ and can be traversed any number of times at cost $c(u)$ (e.g., a duration or distance). The depot is represented as a loop σ. A subset R of nra arcs require service by a vehicle. Each required arc u has a demand $r(u)$ and a service cost $w(u)$. All costs and demands are non-negative integers. The arcs in R represent nt tasks for the vehicles: net edge-tasks (pairs of opposite arcs of R, serviced together, in any direction) and nat arc-tasks (independent arcs of R). Hence, $nra=nat+2.net$. Two arcs u, v for the same edge are linked with a pointer inv such that $inv(u)=v$ and $inv(v)=u$. Moreover, $r(u)=r(v)$ is the total demand on the edge and $w(u)=w(v)$. By convention, we set $r(u)$ and $c(u)$ to 0 if u is not required. This allows for instance to get the total demand by a simple sum over A. We also set $inv(u)$ to 0 if u does not belong to an edge, whether u is required or not.

The graph is defined by giving for each arc u a list $succ(u)$ of allowed successor-arcs: $v \in succ(u)$ if arcs u and v are consecutive ($e(u)=b(v)$) and if it is permitted to turn from u to v. The shortest paths are given by a matrix D, $nra \times nra$. For any two arcs u and v, $d(u,v)$ is the traversal cost of a shortest path from u to v (not included, to ease operations on trips like arc insertions), taking prohibited turns into account. Paths from or to the depot are handled by including a fictitious loop in A for the depot. D can be computed by adapting Dijkstra's shortest path algorithm (Cormen, *et al*., 1990). A trip can be stored as a list of required arcs, assuming shortest paths between successive tasks and between a task and the depot loop. Prohibited turns become transparent.

2.2 A Memetic Algorithm for the Extended CARP

The Memetic Algorithm (*MA*) of (Lacomme, *et al*., 2001) is briefly recalled here, because its

chromosomes and evaluation procedure are partly reused in section 5 for the *CARPTW*. Consider one solution to the extended *CARP*. Its chromosome is built by concatenating the list of tasks of each trip. This gives a sequence of *nt* required tasks in which each task occurs once (an edge can occur as one of its two opposite arcs).

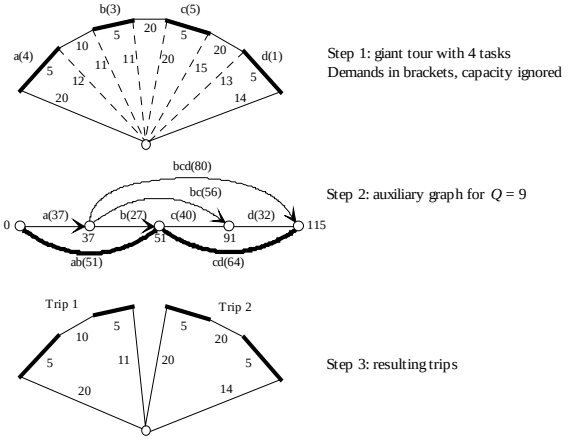

Fig. 1. Evaluation Procedure SPLIT

It can be interpreted as a long tour performed by one single vehicle of infinite capacity, servicing all tasks. This coding is appealing because there always exists one optimal sequence. Chromosomes have no trip delimiters. The trips and their total cost are computed by an optimal method *SPLIT* (Fig. 1). The upper part shows a long tour of 4 (thick) edge-tasks *a*, *b*, *c*, *d* in an undirected context, with demands in brackets. Traversal and service costs are equal. Thin lines are possible shortest paths linking successive tasks or a task and the depot. In the middle, an auxiliary acyclic graph *H* is built with one arc per possible trip, assuming a vehicle capacity *Q*=9: e.g., arc *ab* stands for a trip servicing *a*, *b* and costing 51. A shortest path in *H* (thick line) can be computed using Bellman's algorithm (Cormen, *et al*., 1990). It gives the optimal splitting for the chromosome (here two trips, for a total cost 115).

Thanks to the chromosomes without trip limits, classical crossovers like *OX* and *LOX* can be applied to make children that can be evaluated optimally by the splitting procedure. Searching the best sequence is left to the intrinsic parallelism of the *MA*. The reader will find a description of the other *MA* components in (Lacomme, *et al*., 2001). This *MA* is currently the most efficient solution method for the *CARP*. Solution quality can be evaluated with a tight lower bound proposed by (Belenguer and Benavent, 1997). For instance, on a set of 34 basic *CARP* benchmarks, 22 are solved to optimality (the bound is reached) and the average distance to the bound is only 0.69%. In comparison, the tabu search from (Hertz, *et al*., 2000)) finds the optimum for 15 instances, with an average deviation of 1.9%. The excellent behavior of the *MA* was the main reason to envisage memetic algorithms for solving arc routing problems with time windows.

3. PROBLEM FORMULATION

This section describes the notation and features that are common through this paper. The time window constraint is denoted by a predefined time interval for each task *u*, given an earliest arrival time *et(u)* and latest arrival time *lt(u)*. The vehicles must arrive at the tasks at *at(u)* not later than the latest arrival time, if vehicles arrive earlier than the earliest arrival time, waiting occurs. Vehicles are also supposed to complete there individual routes within a total route time, which is essentially the time window of the depot [$et(\sigma)$, $lt(\sigma)$]. The time windows are two-sided, meaning a task must be serviced at or after its earliest time windows. If a vehicle reaches a task *u* before the earliest time it results in idle or waiting time *wt(u) = et(u)-at(u)*. A vehicle that reaches a task after the latest time is tardy. The objective is to minimize the total vehicle time subject to vehicle capacity, travel time and arrival time feasibility constraint. A feasible solution for the *CARPTW* services all the task without the vehicle exceeding the maximum capacity of the vehicle or the travel time of the vehicle. The total cost is the total travel time for a vehicle which is sum total of the distance travelled by the vehicle including the waiting and service time. Waiting time is the amount of time that a vehicle has to wait if it arrives at task location before the earliest arrival time for that task.

4. CONSTRUCTIVE METHODS FOR THE CARPTW

Two constructive methods were developed. The first one *INSERT1* starts by sorting the task in increasing latest time order in a list *L*. Initially there is an empty trip as a loop on the depot. During the algorithm, such a loop will always be booked after the actual trips, to guarantee the existence of at least one feasible insertion for each task. For each task *u* in *L*, the algorithm tries to insert the task in the current best position according to time windows and distances. The cost variations of an insertion of *u* in all position and in all trips (also in the empty one) are evaluated. The algorithm performs the best insertion.

The second method *INSERT2* is a best insertion approach which has no priority. The method tries to insert the tasks between all the edges in the current route. It selects the task that has the lowest additional insertion cost. The feasibility check tests all the constraints including time windows and load capacity. Only feasible insertions will be accepted. The insertion cost of unrouted task is defined as the weighted combination on additional detour and waiting time needed to insert a task at its best feasible insertion position in the route. The tasks farthest away from the depot are usually the most difficult ones to route, since there are often only a few feasible insertion places available for them. Therefore, the selection of these distant tasks is favored by subtracting from the insertion cost the distance of the corresponding tasks to the depot multiplied by a user defined parameters β. When the current route is full, the heuristic will start a new

route and repeat the procedure until all the tasks are routed.

5. MEMETIC ALGORITHMS FOR THE CARPTW

5.1 Chromosome Structure

Any chromosome *S* for a *CARPTW* solution is then a sequence of *nt* tasks. Each task *u* appears one times, as *u* or *inv(u)*. The sequence can be viewed as a priority list for one single vehicle performing all tasks. This coding is simple but pertinent, since at least one optimal sequence exists. Indeed, any *CARPTW* solution can be defined as one set of trips. We do not put any bit in the string to indicate the end of a route, because such delimiters in a chromosome greatly retrain the validity of children produced by crossover operations later. To decode the chromosome into route configurations, we use the *SPLIT* procedure (see Sect. 2.2). The chromosome can be only decoded to one solution.

5.2 Chromosome Evaluation

The idea is to evaluate the subsequence of tasks with the splitting procedure described in section 2.2 for the *CARP*. Recall that this procedure indicates where to split the subsequence into trips at minimum cost. The cost of a trip *SS* noted *f_trip(SS)* cumulates the collecting costs of its tasks and the traversal costs of its intermediate paths and a penalisation of waiting time. The total cost of a solution *S* noted *f(S)* is the sum of all trip costs. Recall that in formula (1) and (2), *nbtrip* means number of trips after applying the *SPLIT* procedure and $u = 0$ indicate a depot loop σ. As in our encoding we stock only the required tasks, *suiv(u)* represents the task to visit after *u* in *S*.

$$f(S) = \sum_{ss=1\ldots nbtrip} (f\_trip(SS)) \,. \tag{1}$$

$$f\_trip(SS) = \sum_{u=0\ldots nt}[w(u) + d(u, suiv(u))] + \alpha \times \sum_{u=0\ldots nt} Max[0, (et(u) - ar(t))]. \tag{2}$$

5.3 Initial Population

The population is a table *Pop* of *nc* chromosomes whose costs are always distinct. This avoids a premature convergence caused by the generation of identical solutions ("clones") and favors a better dispersal of solutions. *Pop* includes the two constructive heuristics *INSERT1* and *INSERT2* from section 4. We create an initial population in relation to the solution of these two heuristics. The way to do that is by letting the solution of *INSERT1* and *INSERT2* describes a portion of the starting population. The rest of the population is generated on a totally random basis.

5.4 Crossover

This is the most complicated component of the memetic algorithm for the *CARPTW*.
For the proposed memetic algorithm, three crossover operators were proposed: *Crossover1* uses the principle of *LOX* (*Linear Order Crossover*). *Crossover2* is a heuristic crossover which takes into account the time constraint described below and *LOX-T* is a combination of *Crossover1* and *Crossover2*.

Crossover1. The classical *LOX* crossover is well known for permutation chromosomes. It partly transmits the order of elements from the parents to their children. Consider two parents *P1* and *P2* of *n* elements (permutations of the integers 1 to *n*). The construction of the first child *C1* by *LOX* is as follows (the other child is obtained by exchanging the roles of *P1* and *P2*):

Algorithm 1 LOX crossover

draw *a* and *b* with $1 \le a \le b \le nt$
copy *P1(a)…P1(b)* into *C1(a)…C1(b)*
j := 0
for *i* := 1 to *nt*
 if *P2(i)* not already in *C1* then
  j := *j*+1
  if *j* = *a* then *j* := *b*+1 endif
  C1(j) := *P2(i)*
 endif
endfor.

So, *LOX* draws a substring in *P1* and copies it at the same locations into *C1*. Then *P2* is scanned from left to right. Its elements not yet present in *C1* are copied to fill the empty positions of *C1*, from left to right too. *Crossover1* uses *LOX* with additional reparation *Repart_Time (S)* which pushes if necessary the arrival time corresponding to a time windows constraint.

Crossover2. A random cut is made on two chromosomes. From the task after the cut point, the shorter between the two tasks leaving the task is chosen. The process is continued until all positions in the chromosomes have been considered. Suppose, we have the following parents:

Parent 1: 1 2 3 4 5 6 7 8 9
Parent 2 : 5 6 3 1 2 4 9 8 7

Assume we choose 6 to be the first gene in the new chromosome; we have to first swap 6 and 4 in Parent 2. After swapping, if $d(6, 7) + \alpha \times wt(7) > d(6, 9) + \alpha \times wt(9)$, we then choose 9 to be the next task and swap 7 and 9 in the first parent. This process is continued until a new chromosome of the same length and comprising all elements are formed.

LOX-T. Inspired by the fact that both location sequences and time sequences are important in arc routing, we decide to combine the two crossover operators to have a new operator namely *LOX-T* (*Linear Order Crossover with Time Constraint*) so that two parents produce two children by *Crossover1* and then we apply *Crossover2* to the two children to generate one new child.

5.5 Mutation as a Local Search

A local search is applied with a given probability. So, at the beginning of the local search, the input encoded solution is evaluated and converted into a *CARPTW* solution with detailed trips. We propose to evaluate two local search methods used as mutation operator in our memetic algorithm. The first one is based on *Simple Moves* (*SM*) and the second one is *λ-interchange* local search.

Simple Moves Local Search (SM). Each iteration of the local search scans all pairs of tasks *(u,v)* to evaluate the following moves:

- Flip *u* in its trip, i. e. replace *u* by *inv(u)*.
- Move task *u* after *v* (*v* may be the depot loop for this move)
- Move *u* and its successor on the trip after *v* (*v* may be the depot loop for this move)
- Permute *u* and *v*.

The two possible directions are evaluated when a task is inserted to a new position. In the three last moves, *u* and *v* may belong to the same trip or distinct trips. Each neighbourhood exploration stops at the first improving move detected. The whole search stops when no improving move can be found.

λ *Interchange Local Search*. A λ-interchange generation mechanism was introduced by (Osman and Christofides, 1989) for the capacitated clustering problem. It is based on customer interchange between sets of vehicle routes and has been successfully implemented with a special data structure to other problems. The local search procedure is conducted by interchanging customer nodes between routes. For a chosen pair of routes, the searching order for the customers to be interchanged needs to be defined, either systematically or randomly. In this paper, we only consider the cases $\lambda = 2$, which means that maximal two customer tasks may be interchanged between routes. Based on the number of $\lambda = 2$. There are totally eight interchange operators to be defined: (0,1), (1,0), (1,1), (0,2), (2,0), (2,1), (1,2), (2,2). The operator (1,2) on a route pair (*SSp, SSq*) indicates a shift of two tasks from *SSq* to *SSp* and a shift of one customer from *SSp* to *SSq*. The other operators are defined similarly. For a given operator, the tasks are considered sequentially along the routes. In both the shift and interchange process, only improved solutions are accepted if the move results in the reduction of the total cost. We use the first best move that results in a decrease in cost.

5.7 Other MA Ingredients

The incremental replacement is used. At each iteration, two parents *P1* and *P2* are randomly selected in *Pop* by a binary tournament method: two chromosomes are randomly selected and the best becomes *P1*, the same procedure is repeated to get *P2*. *Crossover1* or *Crossover2* or *LOX-T* are applied to *P1* and *P2* to generate two children with *Crossover1* or one child with *Crossover2* or *LOX-T*. If two children *C1* and *C2* are generated, one child is discarded at random. The remaining child may undergo mutation, in fact one of the two local search described in 5.5. If a duplicate cost occurs, the child is rejected. If not, it finally replaces in *Pop* a chromosome drawn above the median cost. The population of distinct solutions must be relatively small (*nc*=30 to 40) to avoid excessive rejection rates.

The *MA* stops after a maximum number of iterations or a maximum number of iterations without improving the best solution

Table 1 Overall comparison of the five algorithms on the 23 CARPTW instances

	INSERT1	*INSERT2*	*MA*1	*MA*2	*MA*3
Avg. cost	646	431	360	323	259
Avg. time (mn)	____	____	5.28	5.4	7.15

6. COMPUTATIONAL RESULTS

All algorithmic components are implemented in Delphi and tested on a 1,4 GHz PC under Windows XP. Since the *CARPTW* is a new problem, we have generated our own files by adding time windows in classical benchmarks for the basic *CARP* (the 23 gdb files, which can be downloaded from (Belenguer, 1997)). Table 1 shows average values of solution cost, running time in minutes for five algorithms: the two heuristics (*INSERT1* and *INSERT2*) and the three memetic algorithms (*MA1*, *MA2*, *MA3*) which uses respectively *Crossover1*, *Crossover2* and *LOX-T* as crossover operator. The mutation operator selected for the three memetic algorithms is *Random(SM, 2-interchange)* which chooses *SM* or *2-interchange* at random.

The results indicate that the memetic algorithms bring important savings (16% for *MA1*, 25% for *MA2* and 40% for *MA3*) compared to the best constructive heuristic *INSERT2*. The *MA3* algorithm (the memetic algorithm with *LOX-T* crossover) is more effective while requiring comparable computational times.

7. CONCLUDING REMARKS

In our knowledge, this paper is the first one dedicated to the arc routing with time windows. As said in introduction, the proposed memetic algorithms are not restricted to the undirected *CARPTW*: they can tackle extensions of *ECARP* like forbidden turns, mixed graphs, etc.

This research shows that memetic algorithm can obtain good solutions to arc routing problems with time windows compared to insertion heuristics. Results show the efficiency of the new crossover operator *LOX-T* to tackle the time windows constraints in arc routing.

Further research is required on the *CARPTW*, for instance there is clearly a need for a good lower bound to evaluate the quality of heuristic solutions and more appropriate instances must be developed to confirm the results. The problem will be enriched by considering time windows that define prohibitions of access on arcs, which is crucial to solving real-life problems, like the ones encountered in urban waste collection.

REFERENCES

Belenguer, J. M, (1997). Directory of CARP instances. *http://www.uv.es/~belenguer/carp.html*.

Belenguer, J. M. and E. Benavent (1997). *A cutting plane algorithm for the capacitated arc routing problem*, Research Report, Dept. of Statistics & OR, Univ. of Valencia, Spain.

Beullens, P., L. Muyldermans, D. Cattrysse and A. Van Oudheusden, (2001). *A guided local search heuristic for the Capacitated Arc Routing Problem*. Working Paper 01/20, Centre for Industrial Management, KU Leuven, Belgium (to appear in EJOR).

Chiang, W. C. and R. Russel (1996). Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Ann. Operat. Res*, **63**, 3-27

Cormen, T. H., C. E. Leiserson and R. L. Rivest, (1990). *Introduction to algorithms*. The MIT Press.

Fleury, G., P. Lacomme, C Prins and W. Ramdane-Cherif (2005). Improving robustness of solutions to arc routing problems. *JORS*, **56 (5)**, 526-538.

Golden, B.L., J. S. DeArmon and E. K. Baker (1983). Computational experiments with algorithms for a class of routing problems'. *Computers and Opet. Res*, **10(1)**, 47–59.

Golden, B.L., R. T Wong (1981). Capacitated arc routing problems. *Networks*, **11**, 305–315.

Hertz, A., G. Laporte and M. Mittaz (2000). A tabu search heuristic for the capacitated arc routing problem. *Oper. Res*. **48(1)**, 129–135.

Hertz, A., M. Mittaz (2001). A variable neighbourhood descent algorithm for the undirected capacitated arc routing problem. *Transportation science*, **35**, 425-434.

Hirabayashi, R., R. Saruwatari and N. Nishida. (1992). Tour construction algorithm for the Capacitated Arc Routing Problem. *Asia-Pacific J. of Oper. Res*, **9**, 155–175

Lacomme, P., C. Prins and W. Ramdane-Cherif (2001). A genetic algorithm for the capacitated arc routing problem and its extensions. In: *Applications of evolutionnary computing* (E.J.W. Boers, Ed.), Lecture Notes in Computer Sciences, 2037, pp. 473–483, Springer.

Lacomme, P., C. Prins and W. Ramdane-Cherif (2004). Competitive Memetic Algorithms for Arc Routing Problems. *AOR*, **131 (1-4),** 159-185.

Lacomme, P., C. Prins and W. Ramdane-Cherif. (2005). Evolutionary algorithms for periodic arc routing problems. *EJOR*, **165 (2)**, 535-553.

Osman, IH., N. Christofides (1989). *Simulated annealing and descent algorithms for capacitated clustering problem*. Research report. Imperial College, University of London.

Pearn, W. L., A. Assad and B. L. Golden (1987). Transforming arc routing into node routing problems, *Computers and Oper. Res*, **14**, 285–288.

Kohl, N. (1995). *Exact Methods for Time Constrained Routing and Related Scheduling Problems*, PhD. Institut for Matematisk Modellering, Danmarks Tekniske Universitet.

Schulze, J., T. Fahle (1999). A parallel algorithm for the vehicle routing problem with time window constraints. *Annals of Operat. Res*, **86**, 585-607.

Thangiah, S. R. (1995). Vehicle Routing with Time Windows using Genetic Algorithms. In: *Application handbook of Genetic Algorithm*: NEW frontiers (L. Chambers, Ed.), Vol. 2, pp. 253-277.